VZDU # 32 SEPT/OCT 1991

VZ DOWN UNDER CLUB

Editor
Harry Huggins
12 Thomas Str.
Mitcham. 3132.
03-873 1408

Treasurer
Ron Allen
2 Orlando str.
Hampton.3188.
03-598 4534

FIRSTLY, OUR CONGRATULATIONS TO THE EDITOR OF OUR CONTEMPORARY NEWSLETTER, JOE LEON. HE IS A GRANDFATHER. A GRANDSON.

I hope you all enjoyed our birthday issue. Now I have to get this issue out, and then start on our Xmas issue. Wait for it. It will be a special!

I also hope to see a lot more of our members send in more contributions. If it was'nt for Bob Kitch and David Wood, this issue would be about 4 pages.

It's your club, so let us see your name at the head of a column. Even if it's only a letter to say "why don't you do this?"

One member commented that another was sweeping the High Scores table clean. On checking I found that they have 7 high scores and he had 15. Who is kidding who?.

There are still a lot of scoring games not in that high list. Three have been added this issue, but there are many more. A score of 5 could get you in the table. Why not try it? Merkfruit Lodge has a scoring system, so has Chess. Also Ace of Aces, Armada, and many more. I don't mind filling the whole page with scores.

What I would like is some small programs sent in. It does'nt matter how simple. I often have a half page to fill and I have to write some nonsense to save having it blank. Some of you youngsters (70 plus) could fill that in nicely.

Also some more comments/suggestions for the Graphics Competition please.

And for local members,remember that our meetings are the first Sunday of each month, including the 5th of January, 1992. Also on the 13th October there is a buy-sell-swap on at the Box Hill Town Hall. We are aiming to have a stand there. If you get there look us up. You may use it if you have any gear you want to sell. No charge.

ASSEMBLY LANGUAGE

— A Coward's Introduction.

by Bob Kitch

Assembly Language programming can accomplish miracles, but it is made to seem almost as mysterious as Babylonian hieroglyphics ! Most users are fairly well versed in Basic programming but do not often venture into the realms of Assembly Language. This is a pity, as Assembler can achieve things that Basic has never dreamed of.

One of the nice things about Basic is that you can see the results straight away. Type in PRINT"HELLO", press <RETURN>, and the VZ displays "HELLO" on the screen. This is great encouragement to go on from. Assembler, on the other hand, has always been presented as a massive task, requiring the mastering of strange registers, indirect addressing and other things that boggle the mind of a Basic programmer. If only you could do something simple with Assembler that would show results right away.

This article is about starting off slowly with Assembler - and achieving some nice small things wih it. It also builds upon my FAST BASIC theme and some powerful applications have been presented in that hybrid language. Some of the programs written and presented in user group magazines include, REAL TIME CLOCK, SCREEN PRESERVER, LIVENUP and TONE GENERATOR. We will develop some FAST BASIC listings and gain some insight into Z80 Assembly Language if you want to.

1. DOING IT FROM BASIC.

Before we start programming, it is necessary to pose a problem! Let's do something in hi-res graphics (MODE(1)) because this is a little challenging. A commonly required problem is to initialize the hi-res screen in one of the four colours that it is capable of displaying. Normally from Basic, the screen comes up in the default colours of green or buff, depending upon the background value (B = 0 or 1) that is set by the COLOR,B command. Some applications look better if the hi-res screen is yellow/cyan, blue/magenta or red/orange when initialized. The CLS command cannot set-up the screen in these alternate colours. We will first of all write a program using standard Basic commands to achieve a blue hi-res screen.

Listing 1 is called SNAIL GRAPHICS (for obvious reasons). Enter the program and run it. All of the commands should be readily understandable. If they are not, then you are probably not yet sufficiently experienced to proceed to the next step.

SNAIL GRAPHICS Version 1.1 uses a double nested loop and the SET command to fill the hi-res screen. The colour of the screen can be altered by changing the foreground colour in line #130. The two SOUND commands are used so that the time taken to fill the screen can be measured with a stop watch. Isn't it painful! It takes almost one minute to do it. This is much too slow for such a simple exercise.

Observe that the use of the SET command requires that all screen pixels, in the range of 0 to 63 and 0 to 127 in the vertical and horizontal directions respectively, be SET. The colour of the pixel is determined in the COLOR command. No knowledge of the memory locations of the screen (video RAM), or the values to which the memory locations are set, to achieve the colour required by the programmer. The

penalty for this simple approach is time.

Let's now investigate how we can speed-up this screen fill problem.

2. STILL IN BASIC.

Take a look at listing 2. To understand SNAIL GRAPHICS Version 2.3 you must have worked with graphics and progressed to PEEKs and POKEs into the video RAM area. Notice that only a single nested loop is required in this version. Also the POKE to the video area in line #210 actually sets four pixels rather than one. The SOUND commands allow timing of the screen fill and a value of around 8.3 seconds should be obtained. This is obviously a pretty souped up version using the Basic language. Some other small changes to the program could get the timing down to about 8.0 seconds - a huge improvement over Version 1.1. But we are really looking for a near instantaneous method, so let's progress to Assembler.

3. THE VISIBLE Z80

I introduce a technique here that I have not seen tried anywhere else, and that is to examine Z80 Assembler as if it resembled Basic. This should make the understanding of Assembler appear as a simpler and more familiar step for programmers. The method was introduced in my program called LOGIC OPERATIONS that has been previously published. In that program Boolean Logic Operators and the Flag Register of the Z80 were simulated. In the next listing I use Basic to simulate the powerful Z80 Block Move instruction.

Before introducing the third listing, I need to diverge to discussing Assembly Language. In all texts and articles introducing Z80 Assembler, the Block Move is introduced as one of the simpler opcodes to comprehend. I agree with this, and it is why I chose to use a screen fill as the programming example in this introduction. I will endeavour to explain the Block Move instruction in plain English so that it can be comprehended. (The Block Move instruction has been described in my hi- and lo-res screen preserver article.)

The Z80 Block Move takes a number of forms but the one we will use has the mnemonic LDIR. What does this gobbly-gook mean? Firstly, LD means LoaD; this implies a transfer of a VALUE from a SOURCE location to a DESTINATION location. Secondly, the I means Increment or Increase by one; implying that a progressive transfer, or filling, of the DESTINATION with the VALUE is to occur. And finally, the R means Repeat for a certain number of times, or COUNT. Does this look as though it is useful to solving our screen fill problem? Let's think a little further. (If you reflect upon Listings 1 and 2, you will see that these four parameters are also required by Basic.)

We certainly need to fill a well-defined area of memory. The video RAM for the hi-res screen is 800H (2048D) bytes long. This suggests COUNT or SIZE. The video RAM commences at address 7000H (28672D) and ends at 77FFH (30719D). Note that this is 800H or 2048D bytes long. (You may need to think about that one!) The starting address of the screen suggests a DESTINATION for the commencement of the Block Fill. The actual VALUE to be placed in video RAM is to be supplied by the user as this determines the colour of the screen - recall listing 2. The actual location of the VALUE can be regarded as the SOURCE for the Block Fill. By Repeatedly LoaDing the SOURCE VALUE to the DESTINATION for a certain COUNT, and Incrementing the DESTINATION, we should be able to fill the video RAM with a particular VALUE. O.K.,

so the LDIR opcode appears as if it will do the job for us.

With that bit of background, you can now examine listing 3. This is a Basic program written for the "VISIBLE Z80". The Basic program simulates what an Assembler program could look like that does a screen fill. The use of variable names are especially chosen to correspond with the "mysterious" register names on the Z80.

By way of further background, the Zilog (the manufacturers of the Z80) specification for the LDIR instruction is as follows -

a. the HL register points to the SOURCE of the LoaD.
b. the DE register points to the DESTINATION of the LoaD.
c. the BC register contains the COUNT or number of times the LoaD is to take place.
d. the A register is the transfer point between the SOURCE (HL) to DESTINATION (DE) locations.
e. after each transfer, the HL and DE registers are Incremented and the BC (COUNTER) register is decremented.
f. the progressive LoaDing is Repeated until the BC register is decremented to zero - in which case the instruction is terminated.

Conditions a. to c. are the initialization for the LDIR instruction. The actual execution of the instruction involves conditions d. to f. The way in which Zilog express all of this is a little cryptic but for those of you interested it is as follows -

(DE)<-(HL), DE<-DE+1, HL<-HL+1, BC<-BC-1, REPEAT until BC=0.

I trust that you can figure some correspondence between this concise description and my "word picture" provided above.

In lines 200 to 460, the Basic program with the corresponding psuedo-assembler is given. Try to follow the relationship between the two languages. Reading and understanding other programmers' code is a very useful form of learning. I mentioned that the Assembler LDIR instruction is quite powerful, it takes 4 Basic commands to execute it! This is unusual, as normally one Assembler instruction will translate to one Basic command. A full discussion of the Assembler routine is given in the next section.

The algorithm used in this example is simple to understand using the characteristics of the LDIR instruction. There are many other routines that would achieve the same job. It is important to realize that there is never merely one way to achieve a particular end. The programmer generally calls upon his experience with familiar commands, and these may be the quickest and shortest way of achieving the object. But there is never only one correct solution to coding a particular problem. Another algorithm that will achieve a screen fill is as follows-

```
   LD HL,7000H
   LD BC, 800H
LP LD (HL),170D
   INC HL
   DEC BC
   LD A,C
   OR B
   JR NZ, LP
   RET
```

So, with this background, listing 3 should be readily comprehensible. The timing is none to startling, but it is a demo program!

4. THE SAME THING IN ASSEMBLER.

The Assembler Code for this exercise would look as follows -

```
LD A, 170D
LD HL, 7000H
LD DE, 7001H
LD BC, 7FFH
LD (HL), A
LDIR
RET
```

How was this program and list of instructions put together? A process called HAND ASSEMBLY was used. (Hand assembly is used for short programs - longer ones are written using the EDASM program.)

To carry out hand assembly, the following pieces of information are required -
-a numeric list of Z80 instructions,
-an alphabetic list of Z80 instructions and,
-a table to convert decimal (0 - 255) numbers to hexadecimal (00H - 0FFH) numbers.

Should any user require suitable copies of the two Z80 code lists, they can send me $2-00 and I will forward copies by return mail. It is an interesting exercise to write a Basic program that will provide a tabulation of dec-hex conversion that is suitable for hand assembly.

Let's examine how this algorithm works. You may wish to refer to listing 3 to gain a complete understanding. The Block Fill is really best thought of as a "ripple-fill". The source is always the byte preceding the destination and as the HL and DE registers are incremented, a destructive overwrite of the previous information in the block occurs. Note that the value, contained in the A register, must be loaded into the first memory location of the fill to initialize the process. This initialization is most often forgotten by beginners. Another common mistake is to set the count in BC, one too high because the first byte is initialized "outside" of the LDIR instruction. I trust that users can follow the logic of this algorithm - if not, go back to listing 3 in Basic.

The next problem in hand assembly is to find the corresponding decimal values to POKE into memory as Machine Language. By using the Z80 instruction lists, mentioned previously, a hex equivalent of the Assembler can be derived. These are converted to decimal values using a conversion table (or a calculator) and entered into DATA statements. It is possible to put hex values into DATA statements and use a hex loader routine to poke the values into the designated area of memory.

The only remaining problem is where to load the M/L? This has been discussed in a previous article on FAST BASIC. For listing 4, I have chosen to simply POKE it into the Free Space List. (This is a "lazy" way, but I will fix it up in the next listing!) The Usr Vector is set to the start of the M/L. Save the program to disk/tape before running the Basic program. Assembler is totally unforgiving if any errors are made and you run the risk of having to re-enter the entire program. (be warned!)

Listing 4 achieves the hi-res screen fill in about 0.1 secs. This is what we were looking for and it is achieved with 15 bytes of M/L. I trust that this has given some new insight into Assembly Language programming and taken some of the mystery from it. This article also provides a painless introduction to Assembly Language programming.

5. MULTIPLE HI & LO RES SCREENS.

I have provided a bonus for those who have persevered thus far! Listing 5 is the logical culmination of the exercise we set about to achieve at the start of this article. It uses a slightly modified form of the routine described above. It permits the value selected to fill the screen to be passed, from the Basic program, to the USR() routine. The technique used has been described in my FAST BASIC article. The integer value contained in the brackets of the USR statement is placed into the Communication Area at address 7921/2H and can be picked up by the M/L program.

There are three further features of this program that I have not yet discussed and are worthwhile to incorporate into Basic-M/L modules.

i. When entering M/L from DATA statements, the values must be exact. Otherwise, the M/L routine is wrong and a computer malfunction will occur. (The same thing happens if the Usr Pointers are not set correctly.) As a check that the DATA statements have been entered correctly from a program listing, it is useful to set up a "checksum" facility. This simply keeps a running total of the data values as they are loaded. Before executing the program, the checksum is compared to a correct value and execution prevented if the two values do not agree. In listing 5, the variable CS% is used in lines 210 to 240. It is compared with the correct value in line 250 and if not correct passes to an error handling routine in line 1000. If the error message is seen, then the DATA statements in lines 410 to 470 need to be checked.

ii. How many program listings have you read that simply list the DATA values to be POKEd into memory with no explanation? Any wonder that beginners have difficulties in understanding! The method of setting out the M/L in "psuedo-assembler" form is to be strongly recommended.

iii. The program places the 16 bytes of M/L into a reserved area of memory created by lowering the Top-Of-Memory. If at any time, you feel as if the M/L routine is not functioning properly, then it is a simple matter to load a disassembler program and decode the TOM area. This ensures that it is correctly loaded, that your decimal values carry out the action that you think they do, and that the area is not being overwritten by some other processes. This is a very powerful form of debugging when developing M/L routines.

```
5 '                    LISTING 1
10 '***SNAIL GRAPHICS DEMO***
20 '***       HI-RES        ***
30 '***    VERSION 1.1      ***
40 '***  R.B.K. 22/5/86     ***
50 '*** EXECUTION TIME 57.6 SECS.
100 'SET TO HI-RES.
120 MODE(1)
130 COLOR 3,0 :'RED ON GREEN
140 SOUND 10,1
200 FOR V=0 TO 63
210     FOR H=0 TO 127
220         SET(H,V)
230     NEXT H
240 NEXT V
250 SOUND 10,1
260 FOR I=0 TO 2000:NEXT I
270 STOP
280 END
```

```
5 '                    LISTING 2
10 '***SNAIL GRAPHICS DEMO***
20 '***       HI-RES        ***
30 '***    VERSION 2.3      ***
40 '***  R.B.K. 22/5/86     ***
50 '*** EXECUTION TIME 8.3 SECS.
100 'SET TO HI-RES.
120 MODE(1)
130 COLOR ,0 :'GREEN BACKGROUND.
140 V%=170:SOUND 10,1:'RED.
200 FOR I%=28672 TO 30719
210     POKE I%,V%
220 NEXT I%
250 SOUND 10,1
260 FOR I=0 TO 2000:NEXT I
270 STOP
280 END
```

LISTING 3

```
10 '*************************
20 '***  VISIBLE Z-80   ***
30 '***   DEMO OF LDIR   ***
40 '*** ASSEMBLER TUTOR  ***
50 '*** BY R.B.K 7/3/91  ***
60 '*************************
70 '*** EXECUTION TIME 70 SECS.
80 '
90 '***DEMONSTRATION OF SCREEN FILL IN BASIC T
HAT EMULATES
100 '***THE LDIR INSTRUCTION OF THE Z80.
110 '***VARIABLES USED RESEMBLE THOSE OF THE Z
80 REGISTER SET.
120 '
130 '
140 MODE(1)                    :'***HI-RES SCREEN.
150 COLOR ,0                   :'***GREEN BACKGROUN
D.
160 SOUND 10,1                 :'***TIMING MARK.
170 '
180 '***ASSEMBLER SIMULATION STARTS HERE.
190 '***INITIALIZE ALL OF THE REGISTERS USED.
200 A%=170                     :'LD A,170    VALUE.
210 HL%=28672                  :'LD HL,7000H SOURCE
.
220 DE%=28673                  :'LD DE,7001H DESTIN
ATION.
230 BC%=2047                   :'LD BC,07FFH COUNT.
280 '
290 '***PUT FIRST VALUE INTO START OF VIDEO RA
M.
300 POKE HL%,A%                :'LD(HL),A
380 '
390 '***CARRY OUT DESTRUCTIVE BLOCK MOVE.
400 '                          :'LDIR
410 POKE DE%,PEEK(HL%)         :'     (DE)  <-  (HL)
420 HL%=HL%+1                  :'      HL   <-  HL+1
430 DE%=DE%+1                  :'      DE   <-  DE+1
440 BC%=BC%-1                  :'      BC   <-  BC-1
450 IF BC%<>0 THEN GOTO 410:'       TEST FOR END
460 '                          :'RET
470 '
490 '***FINISH OFF.
500 SOUND 10,1                 :'***TIMING MARK.
510 FOR I=0 TO 2000:NEXT I
520 STOP
600 END
```

LISTING 4

```
1 '
2 '**************************************
3 '***NEAR-LIGHT-SPEED GRAPHICS DEMO***
4 '***              HI-RES             ***
5 '***           VERSION 1.3           ***
6 '***         R.B.K.  22/5/86         ***
7 '**************************************
8 '*** EXECUTION TIME <0.5 SECS.
9 '***LOAD MACHINE CODE INTO FSL ABOVE BASIC V
LT.
10 FOR I=-28687 TO -28673
20    READ A:POKE I,A
30 NEXT I
39 '
40 DATA 62,170        :'LD A,170      (#170D BLU
E)
41 DATA 33,0,112      :'LD HL,7000H   (#28672D S
TART VIDEO RAM)
42 DATA 17,1,112      :'LD DE,7001H   (#28673D N
EXT)
43 DATA 1,255,7       :'LD BC,07FFH   (#2047D  S
IZE OF VIDEO RAM)
44 DATA 119           :'LD (HL),A
45 DATA 237,176       :'LDIR          (BLOCK LOA
D COMMAND)
46 DATA 201           :'RET
47 '
49 '***INITIALIZE USR() TO ADDRESS 8FF1H OR #-
28687D IN FSL.
50 POKE 30862,241:POKE 30863,143
58 '
59 '***PUT UP BLUE SCREEN.
60 MODE(1):COLOR,0
70 SOUND 10,1
80 X=USR(0)
90 SOUND 10,1
98 '
99 '***DELAY TO VIEW SCREEN.
100 FOR I=0 TO 2000:NEXT I
110 END
```

first loop is the forty tracks, the second the sixteen sectors, the third the five sections that make up the sectors and finally the repitition of values in these sections.

The five sections are composed as follows. The first gap comprises seven bytes of 80H followed by one byte of 0. The identification address mark, also called the IDAM is made up of seven bytes. FEH, E7H, 18H, C3H, the track number, the sector number and a checksum which is calculated from the sum of all the other numbers in the IDAM. If this is not what it should be during a LOAD or SAVE you usually get the dreaded 'SECTOR NOT FOUND' error or the equally bad 'CHECKSUM ERROR'. The second gap consists of five bytes of 80H, and one byte each of 0, C3H, 18H, E7H and FEH. Next is the data field, where your programs go. It is created by simply one hundred and twenty eight zeros. Finally another checksum comprising two bytes is calculated.

Within each of the above mentioned sections of dissassembly listing, not a lot goes on. The timing, which is very critical, is calculated by the actual hardware. All the software (DOS) does is to load the data bit from a list and then send it to the drive. The actual data is recorded using Frequency Modulation (FM). We want to know how this affects the data. Each BIT of data consists of two CELLs on the disk. There is a CLOCK BIT which is one MICRO SECOND long. A micro second is one millionth of a second and is represented by uS. Following this there is an eleven uS gap until the actual bit of our information. If this bit does not exist then it is assumed to be zero. To check this there is a 32.2 uS gap from the start of one clock bit to the start of another clock bit. DUE to the simplicity of FM recording in the VZ drive, The rotation of the disk is slowed from the industry standard of 300 RPM to 80 RPM. This also allows a reduction in components and therfore cost. Just compare the number of ICs in an IBM drive to the six or seven in the VZ drive.

After all that we return to the dissassembly listing to show that address 4CF1-4CF5 steps the head in once to lay down another track. When forty of these have been done the head is once more stepped out forty times (4D03-4D07) and the verification occurs. The track and sector (0,0) are loaded into the track / sector vectors and a search for the IDAM begins (4D10-4D16). If this is not found then we are returned to BASIC with an 'I/O ERROR'. If not then the next is searched for, and so on until all are found. I have only ever had one disk fail during initialisation and this was due to a scratch on the under side of the disk. (Your data is recorded on the opposite of the disk to the label). This scratch was due to the disks previous use in an APPLE disk drive. Anyway, if the disk is okay then the next address after the INIT command was issued is loaded back into the program pointer, the power to the drive turned off and we jump back to BASIC. (4D40-4D44).

Hopefully you are not all confused and now realise how much work is done when you press four keys. I-N-I-T-<RETURN>.

HOW THE VZ INITIALIZES A DISK

By BEN HOBSON

Some time ago Harry asked me to write an article on how the VZ initializes a disk. After a bit of hacking and careful study of dissasembly listings (about 13 pages) her it is. Before we start a few things have to be cleared up. This is not an article which says it writes 0's on the disk. This is an article which explains how the particles are arranged on the disk. While I'm at it, I'll also define a few terms. STEP IN refers to the head moving in one track towards the centre of the disk. STEP OUT is the exact opposite. A disk as you may know is composed of TRACKS and SECTORS. A track is rather like an onion ring. The VZ has 40 of these tracks, one inside each other. The outermost track is called track 0 and the innermost is track 39. In each track there are 16 sectors. A sector is similar to a sector of a circle in mathematics. In mathematics, the sector is a wedge of the circle from the centre to the edge. In computing the sector is similar but is a section of the mathematical sector one track wide. Logic tells us they would be numbered 0 to 15 in a clockwise direction (the way the disk spins). In practise this is not so. The sectors are what is called INTERLEAVED. Interleaving means the next numerical sector is not directly next, but a number of physical sectors away. The reason for this is that computers are generally not fast enough to read the data in the sector (128 bytes in the VZ) and process it. By the time the data has been processed, hopefully the next sector is under the head. If the sectors were sequential the disk would have to rotate once to get the next sector. In the VZ a two sector interleave is used. This means between each numerical sector there is two physical sectors. The order is 0, 11, 6, 1, 12, 7, 2, 13, 8, 3, 14, 9, 4, 15, 10, 5. In the VZ's case this is a little too much. That is why people like Dave Mitchell have written patches for DOS to create a one sector interleave. This considerably speeds things up.

Now for the technical stuff. If you really want to know what is going on a dissassembly listing from 4B08 to 4D44 hex will come in handy. (Two pages)

To start with a DI command is issued (4B08). This dissables the interrupts to ensure the DOS is not disturbed by some trivial thing.

Then the power is turned on to the drive (4B09) via a call to the power on routine. A slight delay is called for to allow the motor to get up to speed (4B0C-4B13). Next the disk is checked to see if a write protect label is in place (4B14-4B19). If so the DOS jumps back to BASIC with the appropriate message.

Next track 0 sector 0 is loaded into the track / sector vectors. The head is then moved out 40 tracks. This achieves two purposes. To make a lot of noise and to ensure the head is over track 0. A small delay is called once more and the action begins.

There are now seven fairly similar looking routines. They are; 4B94-4BBA, 4BBB-4BE1, 4BE2-4C08, 4C09-4C2F, 4C30-4C56, 4C57-4C7D and 4C7E-4CA4. These routines are responsible each for a different value. Each sector consists of five parts. It starts with a gap, then an identification mark, another gap, the actual data field, and finally a checksum. The five numbers needed in a sector are; 0, 80H, FEH, E7H, 18H, C3H and a checksum. The actual procedure of laying down the sectors is a series of loops. The

GAMES COLUMN

Paul has'nt been able to make it this issue again.

Bear with Paul for now. He has some important exams coming up, and no doubt will be on tap for next issue.[Ed].

I have a correction to make. Last month I credited Mathew McLean with Peter Watson's score of 1918 for Super Snake. I have corrected it this issue.

The HIGH SCORES

GAME	SCORE	LEVEL	HOLDER
DAWN PATROL	65500		Martin Wedgwood
CRASH	581		Peter Watson
DIG OUT	52500		Kenley McLean
HAMBURGER SAM	46400		Peter Watson
LADDER CHALLENGE	23970		Peter Watson
KAMIKAZE	113410		Peter Watson
TEN PIN BOWLS	206		Bernice O'Mahoney
VZ INVADERS	30160		Peter Watson
GALAXON	328,460		Mathew McLean
PENGUIN	2320		Peter Watson
LUNAR LANDER	17400		Peter Watson
SUPER SNAKE	1918		Peter Watson
MAZE OF ARGON	78306		Peter Watson
ASTEROIDS	110000		Peter McLean
CIRCUS	2690		Kenley McLean
PANIK	11090		Martin Wedgwood
HOPPY	25550		Matthew McLean
GHOST HUNTER	23400		Chris McLean
STAR BLASTER	480 units left		Matthew McLean
KNIGHTS & DRAGONS	5300	Easy	Peter Watson
KNIGHTS & DRAGONS	1200	Expert	Peter Watson
SPACE RAM	805		Peter Watson
MISSILE ATTACK	21460		Peter Watson
BUST OUT	1970		Peter Watson
PLANET PATROL	471		Peter Watson

OTHER V Z USER GROUPS

H.V.V.Z.U.G
P.O.Box 161
JESMOND NSW.2299.

DISKMAG
P.O.Box 600.
Taree NSW. 2430.

CENT.VIC.COMP.Club
24 Breen St.
BENDIGO VIC 3550

BRISBANE VZUG
63 Tingalpa St.
WYNUM West. Q'ld. 4178

Graeme Bywater
P.O.Box 388
MORLEY W.A. 6062

ADVENTURE GAME WRITING - IMPROVING THE VOCABULARY OF THE GAME

Many commercial adventures on larger computers have vocabularies of several hundred words and enable the player to enter commands using English-type sentences. Some also allowed the player to enter compound commands - commands where more than one action can be carried out - such as GET THE WOODEN STAKE, KILL THE VAMPIRE, THEN ENTER.

There are quite a few problems with the last of these, but it is possible to make the computer understand sentences of a sort. The improved vocabulary is due to more plot, and therefore more words to carry out extra actions, extra words used to hold the sentences together, such as "IN" or "THE", and extra words that mean the same thing, such as "LOOK" and "EXAMINE". Extra words make it more likely that the player will choose the right word first time, but may take up a little extra memory.

Some players prefer to be able to communicate in sentences because it makes them feel more a part of the game as they are "talking" to the computer. However for others, the less typing, the better, so for them the old verb-noun parser is preferable.

In MERKFRUIT LODGE some sort of compromise is achieved. It allows players to enter sentences if they really feel like it, but also allows the two word parser to be used.

The parser assumes the first word of the command is a verb, so this has been placed in V$. The rest of the command is left in T$. (Note that as these lines are from MERKFRUIT, they will not fit into the line numbers of the demo program.)

```
350 IFLEFT$(T$,3)="AT "THENT$=MID$(T$,4)
354 IFLEFT$(T$,3)="IN "THENT$=MID$(T$,4)
356 IFLEFT$(T$,4)="THE "THENT$=MID$(T$,5)
```

These three lines remove the AT or the IN from commands such as LOOK AT THE ROCK or LOOK IN THE CUPBOARD. Next the word "THE" is removed from these commands and also commands such as GET THE AXE. Of corse, if there are no ATs, INs or THEs, nothing is removed. Line 357 removes a full stop from the end of the command if there is one there.

```
357 IFRIGHT$(T$,1)="."THEN T$=LEFT$(T$,LEN(T$)-1)
```

The following lines find out the last word of the command. This is for dealing with situations where the following happens:

Player: GIVE AXE

Computer: GIVE THE AXE TO WHOM?

Player: FRED

The player could instead type GIVE THE AXE TO FRED all in one command (or just GIVE AXE FRED if (s)he felt like it).

```
358 FORI=1TOLEN(T$):IFLEFT$( RIGHT$(T$,I),1)=" "THENK=I:I=LEN(T$)
359 NEXT:IFK>0THENL$=RIGHT$ (T$,K-1):K=0
```

Now the last word is left in L$, and the computer will ignore everything but the first remaining word in T$, which it assumes to be the noun. If there is only one word in T$, L$ is left blank.

The contents of L$ can then be used from within a verb routine. If the player has not typed the second noun the routine should still ask GIVE THE (thing) TO WHOM? as before.

```
1640 IFL$<>""THENT$=L$:GOTO1647
1645 PRINT"GIVE THE ";T$;:INPUT" TO WHOM";T$
```

```
1647 U=1:GOSUB XXX:U=0
```
Replace XXX with the line number of the line where the noun number is determined. The noun number of this second object will be placed in B.

```
'Start of noun sorting routine
FORI=1TO NO:IFMID$(Y$,I*3-2,3) =LEFT$(T$,3)THENB=I:I=NO
NEXT:IFU=1THENRETURN
```

This is the equivalent of lines 320-330 of the demo program. Note the use of the "IFU=1THENRETURN" which does not actually appear in the program.

The major problem with using compound commands is knowing when the entire input must be rejected. Obviously if for some reason the player can't GET THE WOODEN STAKE, it is unfair to expect him to KILL THE VAMPIRE with his bare hands. However you might like to design a routine so that a player can go N,N,E,E,S,W all in one move, or GET THE THICKSHAKE AND THE HAMBURGER. Already some VZ adventures feature the command GET ALL, which enables the player to pick up every visible, gettable object in the room at once.

GRAPHICS

Very few adventures for the VZ contain any graphics at all. ADVENTURE ISLAND is the only one I have come across that is graphics based, although several of the VZ Down Under adventures, contained title screens designed using GRAFSTAR, a utility which was formerly available from VZ Down Under, but I am unsure about its current availability.

Because each hi-res screen takes up 2K of memory, it is not feasible to store whole screens unless you store them on disk and load each one into memory as needed, or load them using 16K blocks into the 64K RAM expansion pack.

To achieve the first method, you will need to write a program to draw your hi-res screen and at the end of it place a BSAVE"filename",7000,77FF command. To restore your screen when needed BLOAD your file using the appropriate filename. Of course you will need to change the screen to MODE(1) before doing this and use a different filename for each screen. See the article in VZDU#6 for more details. For the second method, design each screen and load it into a memory block using Bob Kitch's screen mode mover(VZDU#19- also appeared in LE'VZ). Once each 16K block is full save it to disk using a similar command to that above (Note the HEX numbers) or to tape by changing the start and end of basic pointers (see VZDU#7 or #28). To load the screens in your program, load them into memory and then use the screen mode mover to put them on the screen. You will need to change the line numbers of this routine because they will probably clash with those of your adventure. Also note the use of the CLEAR command in line 3050. In this form it will not work as a subroutine, because amongst other things, the CLEAR command tends to clobber Basic's stack, with the result that the computer forgets where it has to RETURN to at the end of the subroutine. To fix this, replace line 3050 with a RETURN command, and when you wish to call the subroutine use GOSUB3000:CLEAR300:GOSUB3060
This method takes up more effort and memory, but is the only alternative for 64K expansion owners who don't own disks, and saves disk and drive wear and tear for those who do. I haven't gone into

~~disk and drive wear and tear for those who do. I haven't gone~~ into full detail because I do not own either a disk drive or a 64K expansion so am not sure whether I have outlined each method correctly. This discussion is more aimed at giving people ideas on what they could do.

Another possible method is to draw the same general room shape for every room, adding doors where required. From this it would be a matter of adding pictures of gettable objects and some fixed objects in the rooms where they appear (eg. bed, table, window, rope, shovel, ticking parcel, etc.). Hardly 256 colour VGA standard graphics but it is a start. You may like to use mixed text/hi-res graphics using the routine in the April/May edition of VZ USER. This routine enabled four lines of text and 127 by 48 pixel graphics, and is similar to one used by Larry Taylor to create Escape River. Because of the length of the routine I cannot reproduce it here, and because VZ USER is no longer produced, I don't know where you can obtain a copy if you haven't got one. If you do have a copy, remember that the Y part of the graphics ranges from 4 to 51 (not 0 to 47 as you might expect), that you cannot use INPUT statements - to get the player's command you must use the INKEY$ statement, and to save your effort regularly - if you make an error or press BREAK while the program is in split screen mode, you will lose control of the system and have to reset the computer. If you use this feature you will also have to keep your descriptions down to a minimum so everthing fits into four lines. The program below is an example of how you could draw a room. Y values are kept in the range of 4 to 51 in case you decide to use the split screen; if you don't use it you can change the program to make the room a bit bigger. The program makes up a random movement code which is then used to draw doors in these positions. In the program the far wall is said to be north. Lines 240 and 250 draw a barred window in the north wall.

```
90 FORI=1TO4:D$=D$+MID$(STR$( RND(2)-1),2,1):NEXT
98 COLOR1+RND(3)
99 MODE(1)
100 FORX=0TO97:SET(X,51):NEXT
110 FORY=19TO51:SET(0,Y): SET(97,Y):NEXT
120 FORX=30TO97:SET(X,36):NEXT
125 FORY=4TO36:SET(127,Y): SET(30,y):NEXT
130 FORI=1TO30:SET(30-I,36+(I/2)): SET(127-I,36+(I/2))
140 SET(30-I,4+(I/2)):SET (127-I,4+(I/2)):NEXT
150 FORX=30TO127:SET(X,4):NEXT
155 IFLEFT$(D$,1)="1"THEN175
160 FORY=19TO36:SET(68,Y):SET (88,Y):NEXT
170 FORX=68TO88:SET(x,19):NEXT
175 IFMID$(D$,3,1)="1"THEN195
180 FORI=10TO20:SET (30-I,19+(I/2)):NEXT
190 FORY=24TO41:SET(20,Y):SET (10,Y+5):NEXT
195 IFRIGHT$(D$,1)="1"THEN215
200 FORI=10TO20:SET (127-I,19+(I/2)):NEXT
210 FORY=24TO41:SET(117,Y):SET (107,Y+5):NEXT
215 IFMID$(D$,2,1)="1"THEN240
220 FORX=38TO58:SET(X,49): RESET(X,51):NEXT
230 FORY=49TO51:SET(38,Y):SET (58,Y):NEXT
240 FORX=40TO56:SET(X,9):SET (X,19):NEXT:FORX=40TO56STEP4
250 FORY=9TO19:SET(X,Y):NEXT:NEXT
1000 GOTO1000
```

HIDDEN EXITS

These are found in many adventures. The player may knock down a false wall, cut through a dense hedge or clump of ivy, or look in an old barrel against one wall to find a secret room or passageway. In this case you will need to alter movement codes mid-adventure.(Equally you may need to alter it if a path which could originally be travelled suddenly becomes blocked.) Because we have decided to store movement codes in data statements only, this is not the easiest piece of programming. In some cases you may be able to just set up your codes so the player finds when an exit is made west that is not possible to go east again. However for others the program must physically alter itself.

Include the Find Line Number routine that we used temporarily last issue at the start of the program along with the other machine code routines.

eg

```
18 FORI=31290TO31304:READA: POKEI,A:NEXT
22 DATA237,91,33,121,205,44,27, 210,217,30,237,67,33,121,201
```

Where you need to use this feature, your first line should be
POKE30862,58:POKE30863,122

Firstly if the change in movement code is for the room the player is in, you must change the contents of D$ to the new movement code.

Next find the next line number OF A LINE THAT EXISTS after the data statement that contains your movement code. For example, if your movement code was in line 5202, and the line after that was 5210, you would follow in your routine with X=USR(5210)

Depending on which exit you wish to change, you must subtract a certain value from X. For east subtract three; for west, four; for south, five; and for north subtract six.

If you wish to create a new exit, POKE X,48. If you wish to block off an existing exit, POKE X,49

For example, if you wish to change the movement code for line 5202 - assume D$="1110" to create a new exit to the north, you would do the following:

```
D$="0110"
POKE30862,58:POKE30863,122
X=USR(5210)-6
POKEX,48
```

Don't forget to restore the USR pointer to the sound routine by adding
POKE30862,82:POKE30863,121

You might wish to record the change in a flag because if you save your game progress, turn the computer off and reload the game and your progress later on, you will find your new exit will have magically disappeared! After the progress has been reloaded, the program could check any changes that need to be made, and make them again.

Finally if you test drive your program, don't forget to change the line back to its original movement code before you save the program. It would also be a good idea to include a little routine to change everything back if the player QUITs or dies.

RANDOM MOVING ITEMS

These are objects that have the power to move around on their own.

They may be some kind of animal, or a small object that can be blown around in the breeze. They add a little interest to the game because the player never knows where they are going to meet.

In the demo program there is a dog that moves around between three different rooms. This is the code that allows it to do so:

```
480 IFF(30)=1ORF(22)=26THEN500
```

If the dog has decided to follow you, or run away, this section is ignored and skipped.

```
485 U=RND(3):IFF(22)=5ANDU=1THEN F(22)=4:GOTO500
490 IFF(22)=9ANDU=3THENF(22)=4: GOTO500
495 IFF(22)=4ANDU=3THENF(22)=5: GOTO500
497 IFF(22)=4ANDU=1THENF(22)=9
```

F(22) is a flag which contains the location of the dog. It may be that the player can pick up a random moving item (eg a piece of paper blowing around in the breeze). In this case you would use an element of the C array to store the item's location.

In most cases the code is simpler than this, if you decide to have your item move along one particular row or column of your map. In this case it is just a matter of calculating the new position. We will assume the item is C(18) and is in an 8X8 map and can move north and south between rooms 16 and 56

```
IFC(18)=RORC(18)=0ORC(18)=65 THENRETURN
U=RND(3)-2:IF(C(18)=16ANDU=-1) OR(C(18)=56ANDU=1)THENU=-U
C(18)=C(18)+(U*8)
RETURN
```

Obviously if you want your object to move east/west, you should not multiply U by 8. Your object should not be able to walk through walls. Experienced programmers might like to make up a routine to enable the item to move anywhere it likes, provided it looks at the movement codes to see if the move is legal.

This virtually concludes the series. There are one or two more articles to come on tokenised room descriptions and the use of program modules and these will appear later on. Following is a brief summary of the steps involved in writing your own adventure, in the order you might wish to carry them out. The issue of VZDU to refer to is also noted.

Think of something to write about!(#27)
Think of a character, a plot and some puzzles (#27)
Make a list of nouns and verbs for the computer to understand (#27)
Draw a map, and compile your movement codes (#27)
Make up some room descriptions(#27)
Type in the machine code, initialisation and data statements (#29)
Type in the screen display and command parser (#30)
Design your verb subroutines, type them in and Install the ON GOSUB lines (#31)
Add sounds (#29) and other extras
Progress tape save routines (#28) DO LAST!

Good luck writing your adventure program. I hope to see a few efforts published in VZDU magazine or the public domain tape soon!

STOP PRESS STOP PRESS STOP PRESS

They said it would'nt happen, but it has! THE GAMES COLUMN IS BACK!

Actually this month it is really Peter Watson's column. Since he has been the only contributer of late, the above is justifiable. Still as I've said before, if just one person contributes every coupl of months, that is more than enough reason to write this column. (Thanks Peter.(Ed.)) Well with that out of the way let me hand the column over to Peter's hints and comments.

Note there are no adventure game reviews this month as I have run out of adventures to review. If anyone has built a masterpiece thanks to David Wood's tutorials, I would be more than happy to review it.

CRASH. Use the speed button as little as possible, as you run out of fuel much more quickly.

ASTEROIDS. Ignore a "chunk" of asteroid and wait for the UFO to appear. It is worth 1000 points. Once you have destroyed it, wait for another UFO etc. (Patience required here).

LUNAR LANDING. Ignore the 100 and 300 point landing pads. The 500 point pads are as easy to land on. (I can crash anywhere.(Ed.))

As well as the abovehints, Peter has provided us with thefollowing mini reviews of a couple of arcade/action games. Thankyou Peter.

KAMIKAZE. I suppose a few club members will skin me for saying this....but a game of Kamikaze lasts too long. After the first few frames, the level of difficulty remains the same. My high score of 113410 took 1hr 17min., after which I had eye strain!!! (It's an endurance test Peter, not a game of skill Ed.)). I think the game would be better, if say the bombs were faster or there were more of them per frame. I also note that as the Kamikazes built up on the screen, (there can be up to 50 of them), the action slows up noticeably. { Could make the test the time to reach 10,000 or so. Ed.}

Asteroids. The game would be less drawn out if there were no extra lives given every 10000. Also it is very easy, when playing with joysticks, to accidentially hit Hyperspace and reappearin a very dangerous spot. Thanks again to Peter for his contribution to this column, especially since this column writer bacame bogged down with school exams. Once again if you have a review, hints orquestions or high scores etc., please send them into my address below. SEND THEM IN NOW AND KEEP V.Z GAMING THRIVING.

See you next editio

Paul Frantz. 25 Crocker St. KIRWAN. Q'ld. 4817.